

# 6

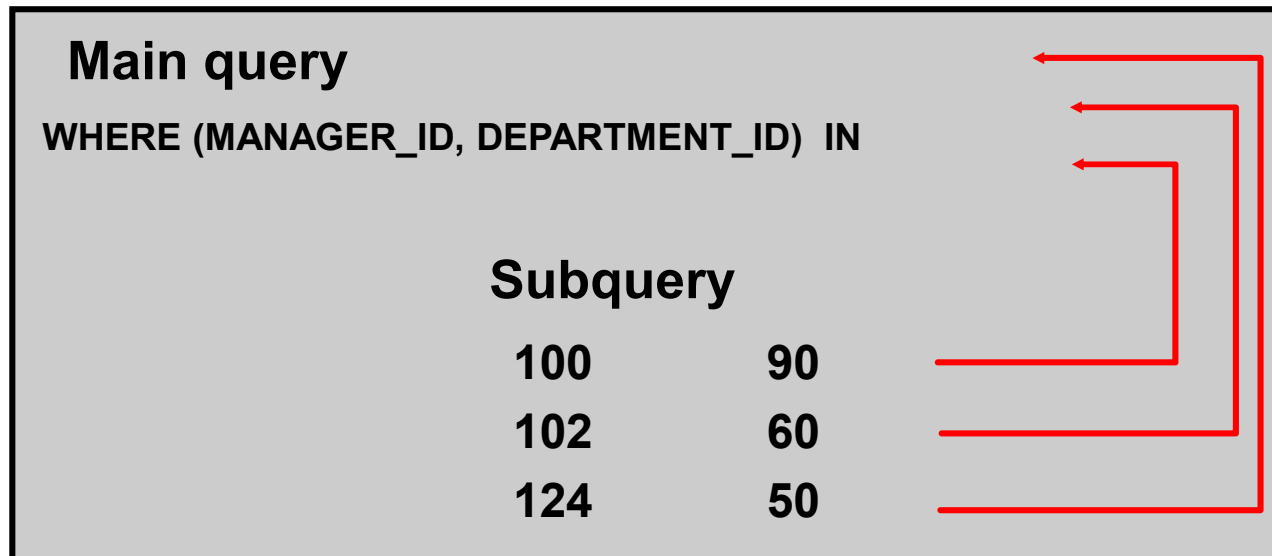
## Alkérdések II.

# Objectives

**After completing this lesson, you should be able to do the following:**

- **Write a multiple-column subquery**
- **Use scalar subqueries in SQL**
- **Solve problems with correlated subqueries**
- **Update and delete rows using correlated subqueries**
- **Use the EXISTS and NOT EXISTS operators**
- **Use the WITH clause**

# Multiple-Column Subqueries



**Each row of the main query is compared to values from a multiple-row and multiple-column subquery.**

# Column Comparisons

**Column comparisons in a multiple-column subquery can be:**

- **Pairwise comparisons**
- **Nonpairwise comparisons**

# Pairwise Comparison Subquery

Display the details of the employees who are managed by the same manager *and* work in the same department as the employees with `EMPLOYEE_ID` 199 or 174.

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE (manager_id, department_id) IN
      (SELECT manager_id, department_id
       FROM employees
       WHERE employee_id IN (199,174))
AND employee_id NOT IN (199,174);
```

# Nonpairwise Comparison Subquery

Display the details of the employees who are managed by the same manager as the employees with `EMPLOYEE_ID` 174 or 199 *and* work in the same department as the employees with `EMPLOYEE_ID` 174 or 199.


```
SELECT  employee_id, manager_id, department_id
FROM    employees
WHERE   manager_id IN
        (SELECT  manager_id
         FROM    employees
         WHERE   employee_id IN (174,199))
AND     department_id IN
        (SELECT  department_id
         FROM    employees
         WHERE   employee_id IN (174,199))
AND     employee_id NOT IN(174,199);
```

# Scalar Subquery Expressions

- **A scalar subquery expression is a subquery that returns exactly one column value from one row.**
- **Scalar subqueries can be used in:**
  - **Condition and expression part of `DECODE` and `CASE`**
  - **All clauses of `SELECT` except `GROUP BY`**

# Scalar Subqueries: Examples

- **Scalar subqueries in CASE expressions**

```
SELECT employee_id, last_name,  
       (CASE  
         WHEN department_id = 20  
           (SELECT department_id  
            FROM departments  
            WHERE location_id = 1800)  
         THEN 'Canada' ELSE 'USA' END) location  
FROM   employees;
```

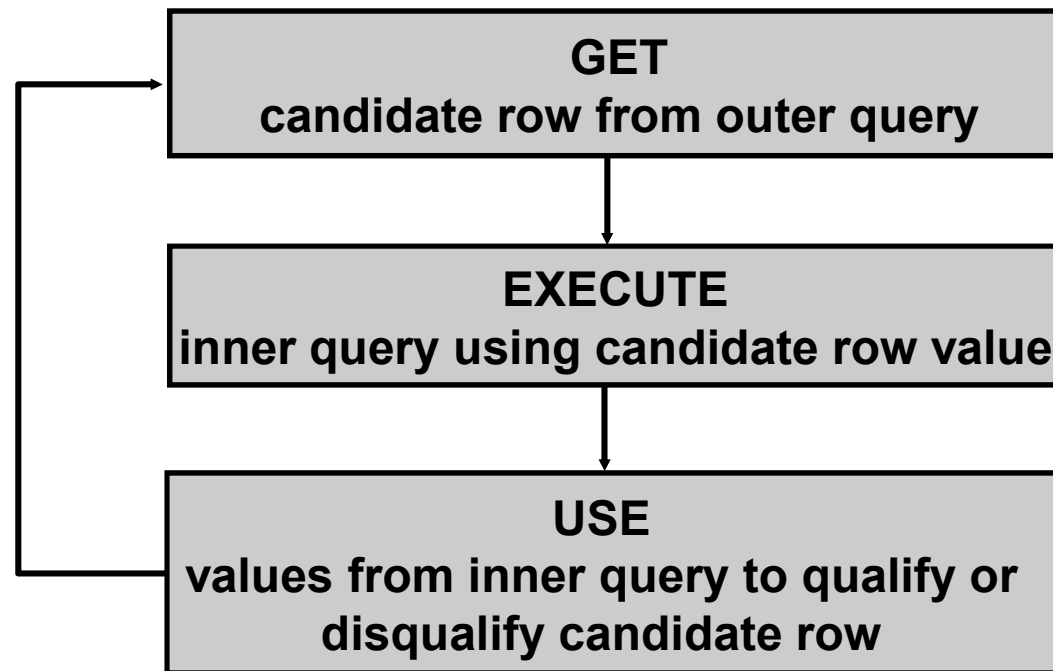
- **Scalar subqueries in ORDER BY clause**

```
SELECT  employee_id, last_name  
FROM    employees e  
ORDER BY (SELECT department_name  
          FROM departments d  
          WHERE e.department_id = d.department_id);
```



# Correlated Subqueries

**Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.**



# Correlated Subqueries


The subquery references a column from a table in the parent query.

```
SELECT column1, column2, ...
FROM   table1 outer
WHERE  column1 operator
           (SELECT column1, column2
            FROM   table2
            WHERE  expr1 =
                   outer.expr2) ;
```

# Using Correlated Subqueries

**Find all employees who earn more than the average salary in their department.**

```
SELECT last_name, salary, department_id
FROM employees outer
WHERE salary >
    (SELECT AVG(salary)
     FROM employees
     WHERE department_id =
     outer.department_id);
```



**Each time a row from the outer query is processed, the inner query is evaluated.**

# Using Correlated Subqueries

**Display details of those employees who have changed jobs at least twice.**

```
SELECT e.employee_id, last_name, e.job_id
FROM   employees e
WHERE  2 <= (SELECT COUNT(*)
             FROM   job_history
             WHERE  employee_id = e.employee_id);
```

EMPLOYEE_ID	LAST_NAME	JOB_ID
101	Kochhar	AD_VP
176	Taylor	SA_REP
200	Whalen	AD_ASST

# Using the EXISTS Operator

- **The EXISTS operator tests for existence of rows in the results set of the subquery.**
- **If a subquery row value is found:**
  - The search does not continue in the inner query
  - The condition is flagged **TRUE**
- **If a subquery row value is not found:**
  - The condition is flagged **FALSE**
  - The search continues in the inner query

# Find Employees Who Have at Least One Person Reporting to Them

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees outer
WHERE  EXISTS ( SELECT 'X'
                FROM   employees
                WHERE  manager_id =
                      outer.employee_id);
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90
103	Hunold	IT_PROG	60
108	Greenberg	FI_MGR	100
114	Raphaely	PU_MAN	30
120	Weiss	ST_MAN	50
121	Fripp	ST_MAN	50
122	Kaufling	ST_MAN	50
123	Vollman	ST_MAN	50
124	Mourgos	ST_MAN	50
145	Russell	SA_MAN	80
146	Partners	SA_MAN	80
147	Errazuriz	SA_MAN	80
148	Cambrault	SA_MAN	80
149	Zlotkey	SA_MAN	80
201	Hartstein	MK_MAN	20
205	Higgins	AC_MGR	110

18 rows selected.

# Find All Departments That Do Not Have Any Employees

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'X'
                  FROM employees
                  WHERE department_id = d.department_id);
```

DEPARTMENT_ID	DEPARTMENT_NAME
120	Treasury
130	Corporate Tax
140	Control And Credit
150	Shareholder Services
160	Benefits
170	Manufacturing
...	
260	Recruiting
270	Payroll

16 rows selected.

# The WITH Clause

- Using the **WITH** clause, you can use the same query block in a **SELECT** statement when it occurs more than once within a complex query.
- The **WITH** clause retrieves the results of a query block and stores it in the user's temporary tablespace.
- The **WITH** clause improves performance.



## **WITH Clause: Example**

**Using the `WITH` clause, write a query to display the department name and total salaries for those departments whose total salary is greater than the average salary across departments.**

# WITH Clause: Example

```
WITH
dept_costs AS (
    SELECT d.department_name, SUM(e.salary) AS dept_total
    FROM employees e JOIN departments d
    ON e.department_id = d.department_id
    GROUP BY d.department_name),
avg_cost AS (
    SELECT SUM(dept_total)/COUNT(*) AS dept_avg
    FROM dept_costs)
SELECT *
FROM dept_costs
WHERE dept_total >
    (SELECT dept_avg
    FROM avg_cost)
ORDER BY department_name;
```

# Summary

**In this lesson, you should have learned the following:**

- **A multiple-column subquery returns more than one column.**
- **Multiple-column comparisons can be pairwise or nonpairwise.**
- **A multiple-column subquery can also be used in the FROM clause of a SELECT statement.**

# Summary

- **Correlated subqueries are useful whenever a subquery must return a different result for each candidate row.**
- **The EXISTS operator is a Boolean operator that tests the presence of a value.**
- **Correlated subqueries can be used with SELECT, UPDATE, and DELETE statements.**
- **You can use the WITH clause to use the same query block in a SELECT statement when it occurs more than once.**